Efficient I/O on the Cray XT

Jeff Larkin
Cray Supercomputing Center of Excellence
larkin@cray.com

With Help Of: Gene Wagenbreth

Overview

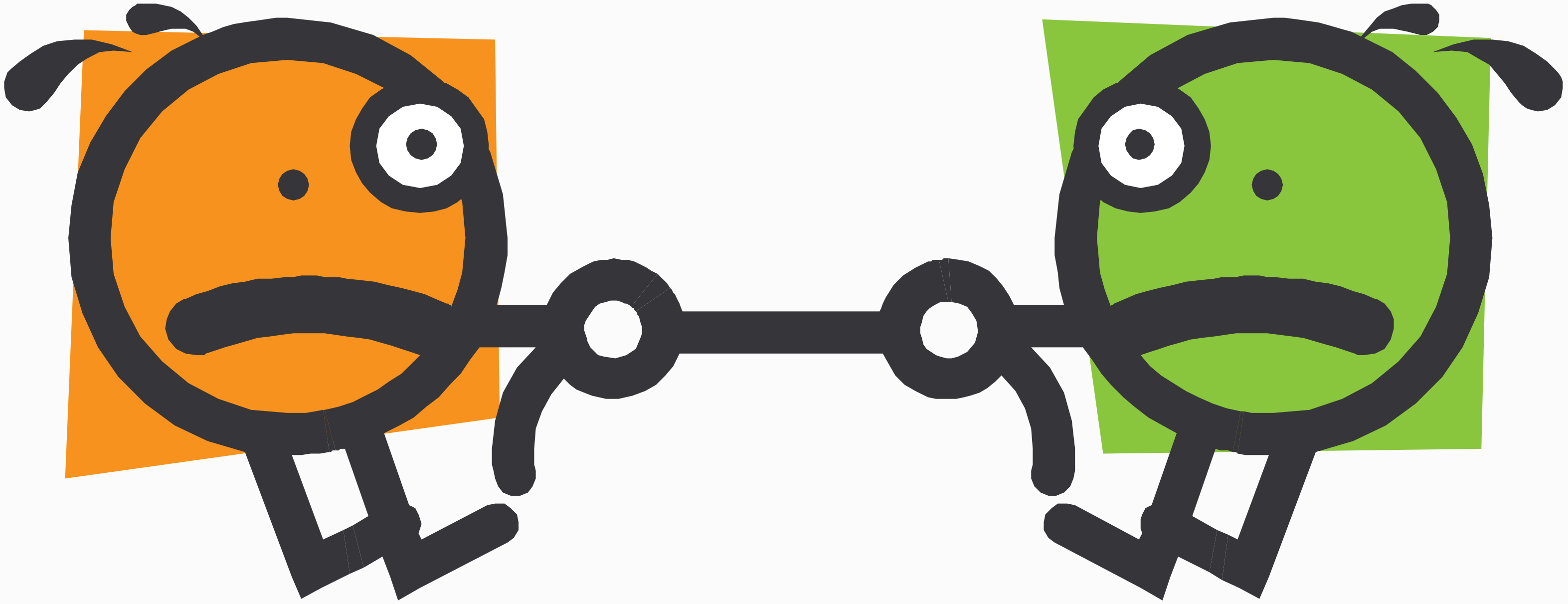
- What's the problem?
- “Typical” Application I/O
- I/O Solutions
- A Solution That Works
- Graphs, so many Graphs
- Take Home Notes

What's The Problem?

- Flops are Cheap, Bandwidth isn’t
- Machines and Applications aren’t getting any smaller
- But...
  - Isn't Lustre enough?
  - Can't I use libraries?
  - Doesn't it just work?
- Without user or programmer intervention, I/O will not perform at peak
- There is no *Silver Bullet*

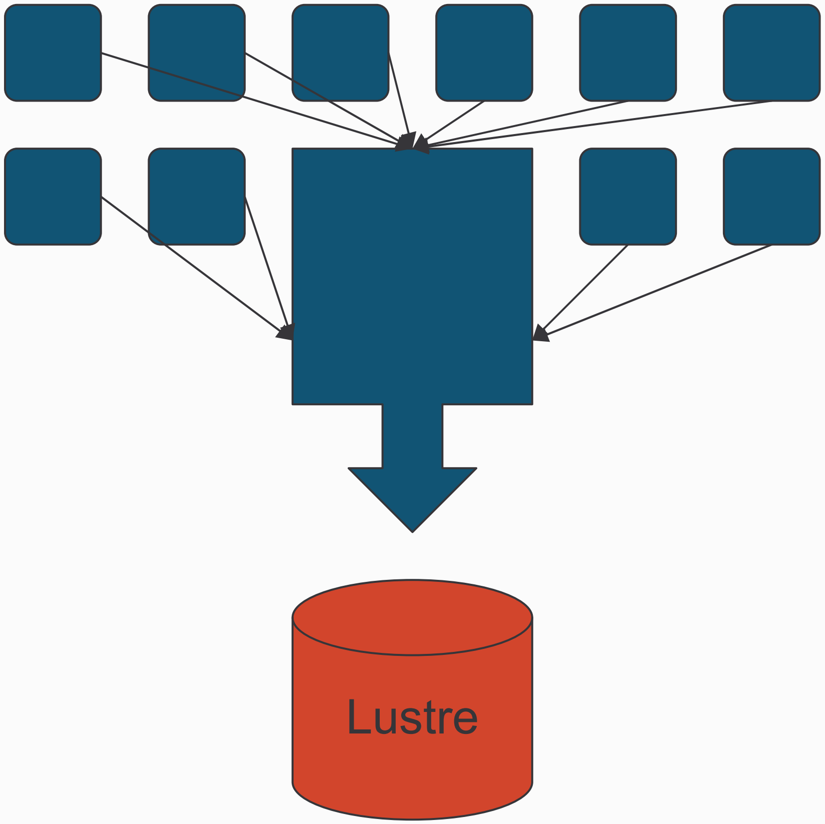
“Typical” Application I/O

- THERE IS NO TYPICAL APPLICATION I/O
- There are several common methods, but 2 are very common and problematic
  - Single-writer reduction
  - N-writer/N-reader to N-files



Simple **Efficient**

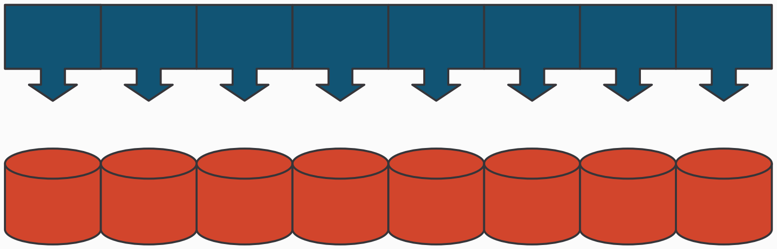
Single-writer Reduction



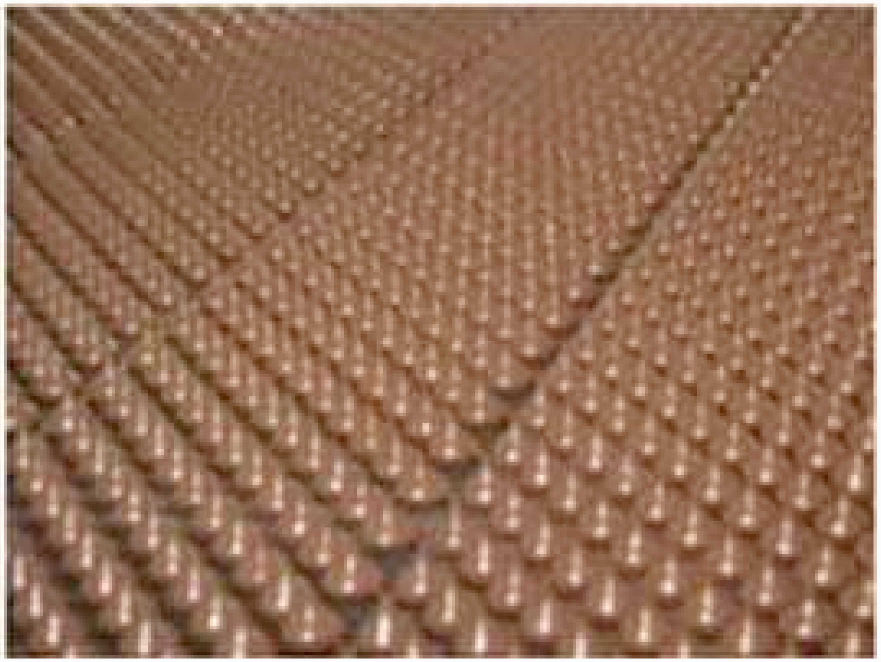
- The Plan
  - All processors send to 1 I/O node for output
  - File striped to maximum OSTs
- The Problem
  - Even with maximum striping, 1 node will never achieve maximum bandwidth
  - single node IO bandwidth is approximately 200 MB/s
  - reading/writing a terabyte would require more than 1 hour at current I/O rates

N-Writer to N-Files

- The Plan
  - Every process opens a file and dumps its data
  - Files striped to 1 OST
- The Problem
  - Can lead to slow opens and general filesystem slowness
  - If the writes are not large, performance will suffer
  - Inconvenient
  - Can only be used as input for same number of nodes
- One Modification
  - Use MPI-I/O for just 1 file
  - Suffers when i/o results in small buffers



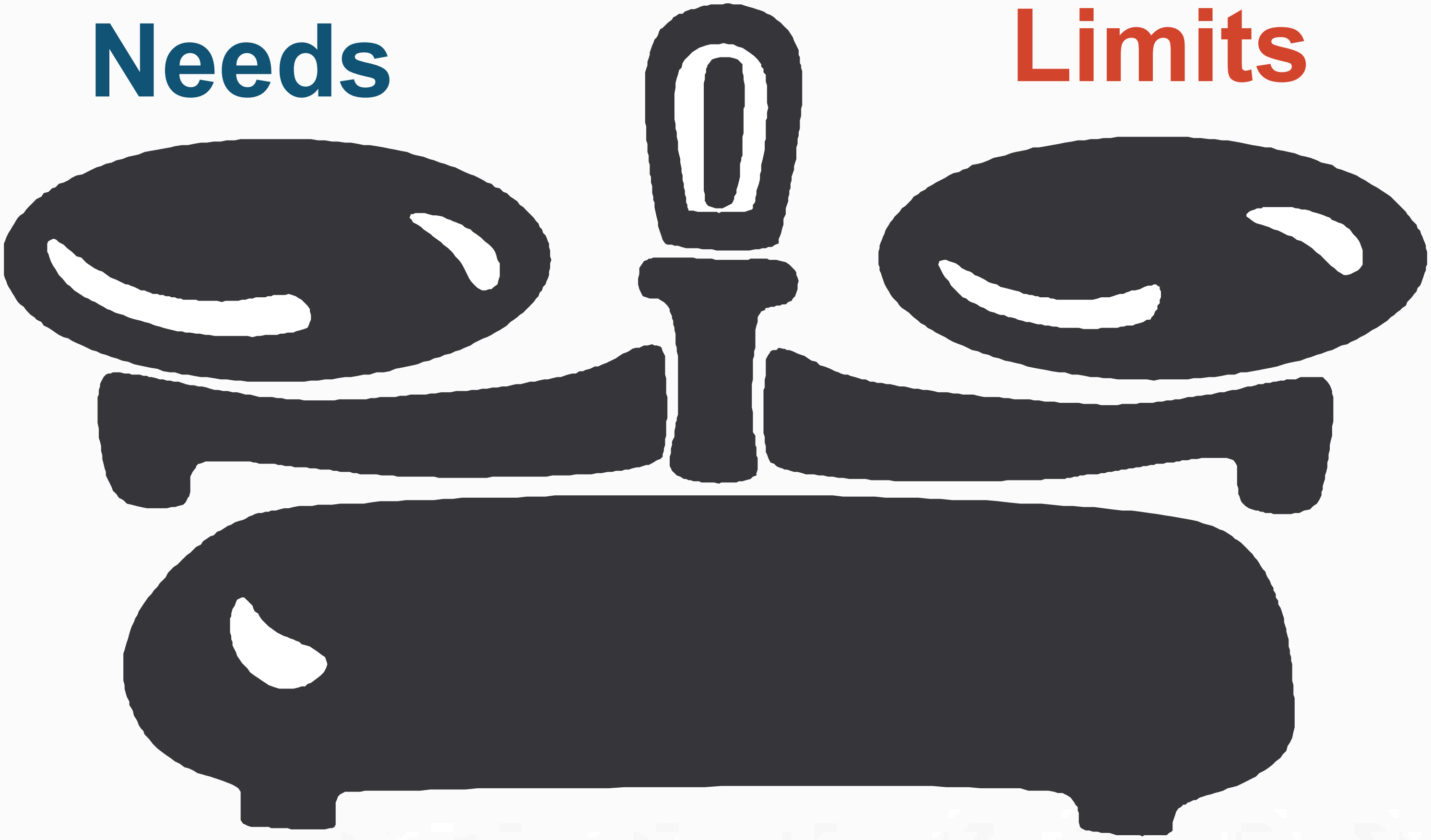
What does efficient I/O look like?



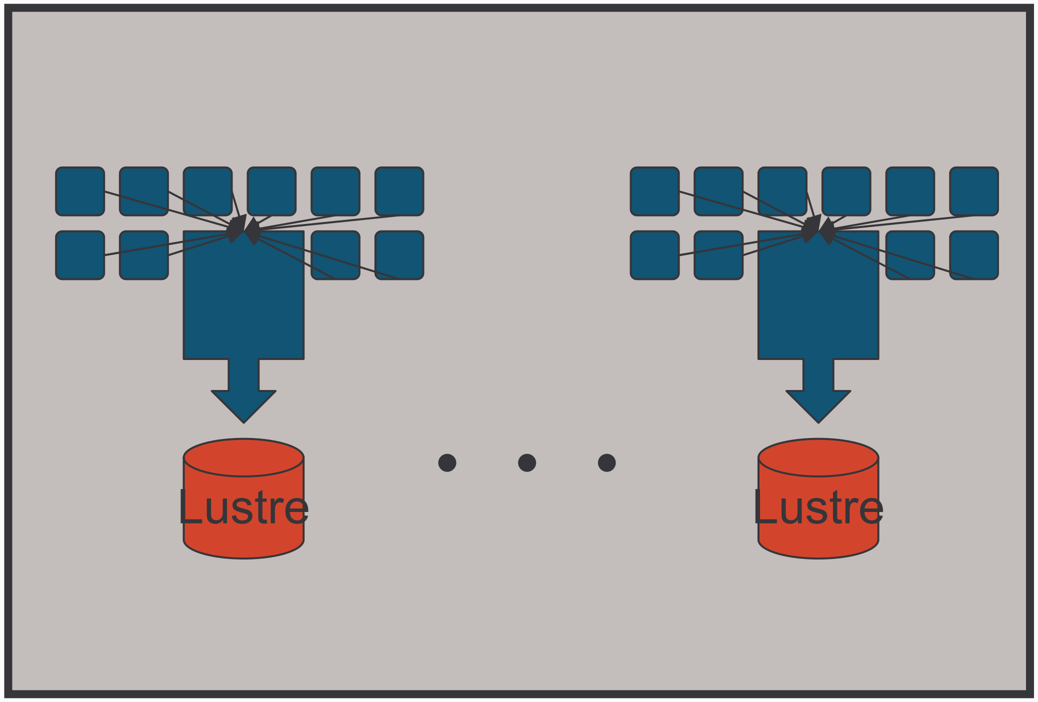
Striking a Balance

Application Needs

Filesystem Limits



Subset of Readers/Writers Approach



- The Plan:
  - Combine the best of our first two I/O methods
  - Choose a subset of nodes to do I/O
  - Send output to or Receive input from 1 node in your subset
- The Benefits
  - I/O Buffering
  - High Bandwidth, Low FS Stress
- The Costs
  - I/O Nodes must sacrifice memory for buffer
  - Requires Code Changes

Subset of Readers/Writers Approach

- Assumes job runs on thousands of nodes
- Assumes job needs to do large I/O
- From data partitioning, identify groups of nodes such that:
  - each node belongs to a single group
  - data in each group is contiguous on disk
  - there are approximately the same number of groups as OSTs
- Pick one node from each group to be the ionode
- Use MPI to transfer data within a group to its ionode
- Each IOnode reads/write shared disk file

Example Code: MPI Subset Communicator

create an MPI communicator that include only ionodes

```
call MPI_COMM_GROUP(MPI_COMM_WORLD,
   WORLD_GROUP,ierr)

call MPI_GROUP_INCL(WORLD_GROUP,niotasks,
  listofiotasks,IO_GROUP,ierr)

call MPI_COMM_CREATE(MPI_COMM_WORLD,IO_GROUP,
  MPI_COMM_IO,ierr)
```

Example Code: MPI I/O

open

```
call MPI_FILE_OPEN(MPI_COMM_IO,trim(filename),
     filemode,finfo,mpifh,ierr)
```

read/write

```
call MPI_FILE_WRITE_AT(mpifh, offset, iobuf,
      bufsize, MPI_REAL8,status,ierr)
```

close

```
call MPI_FILE_CLOSE(mpifh,ierr)
```

Example Code: I/O Code Outline

- IONode:

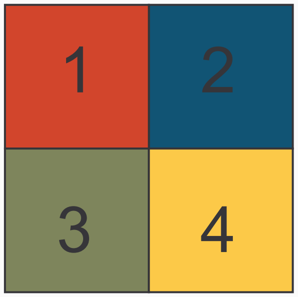
```
copy (scatter) this nodes data to IO buffer
loop over nonIOnodes in this group
mpi_recv data from compute node
copy(scatter) data to IO buffer
write data from IO buffer to disk
```

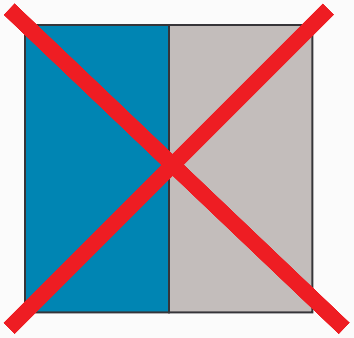
- Non-IONode:

```
copy data to mpi buffer
mpi send data to IO node
```

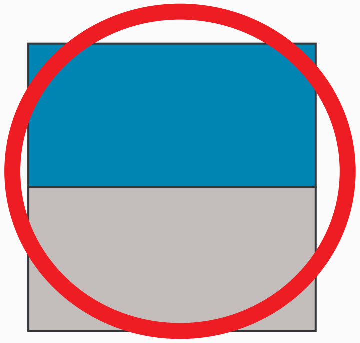
Sample Paritioning: POP

- data is 3d - X, Y, Z
- X and Y dimensions are partitioned in blocks
- sample 4 node partition:
  - Each of the 4 colored blocks represents one node's part of the data
  - Each of the two lighter colored blocks represent 1 I/O Node
  - I/O Groups should be arranged so their data is contiguous on disk





Data from nodes 1 & 3 alternate on disk. This will perform slowly and can't adjust to more processors.



Data from node 1 is contiguous, followed by data from node 2, which is also contiguous.

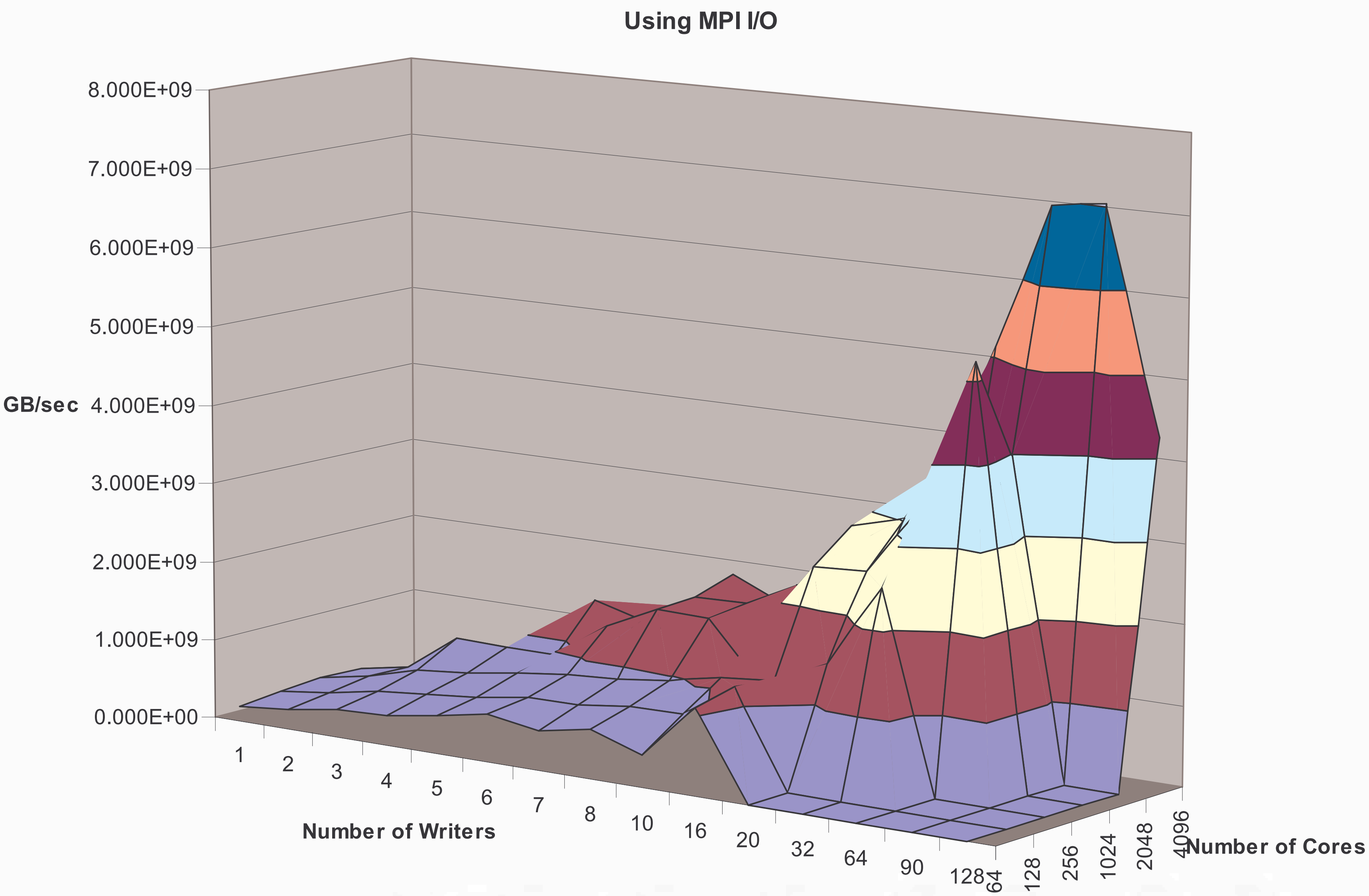
Sample Paritioning: POP

- Given a nearly square partitioning, the number of nodes   simultaneously performing IO is approximately the square root of the total number of compute nodes.
  - 2500 compute nodes - 50 IO nodes
  - 10000 compute nodes - 100IO nodes
  - 25600 compute nodes - 160 IO nodes
- Many partitions allow a reasonable assignment of ionodes

For Example:

- An array of 8 byte reals (300, 400, 40) on each of 10000 nodes
  - 4.8 million elements on each node
  - 48 billion elements total
  - 384 gigabytes data
  - 50 - 100 seconds to read or write at 4 - 8 gbyte/sec
  - 100 IO nodes

A Subset of Writers Benchmark



Benchmark Results: Things to Know

- Uses write_at rather than file partitioning
- Only write data...sorry
  - Read data was largely similar
- Initial benchmarking showed MPI transfers to be marginal, so they were excluded in later benchmarking
- Real Application Data in the works, Come to CUG

Benchmark Results: 1 I/O Node - Stripes

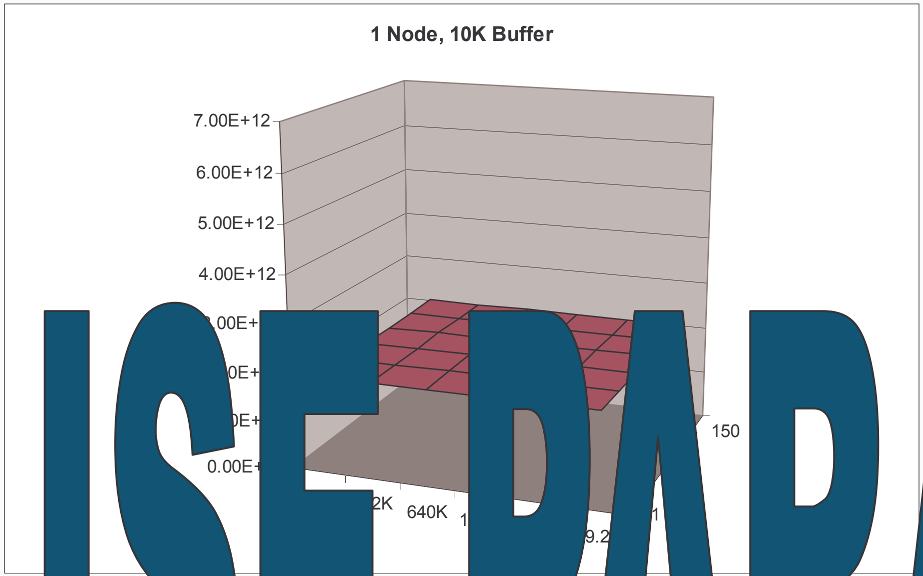
- Single IO node, 10 megabyte buffer, 20 megabyte stripe size: bandwidth of IO write to disk

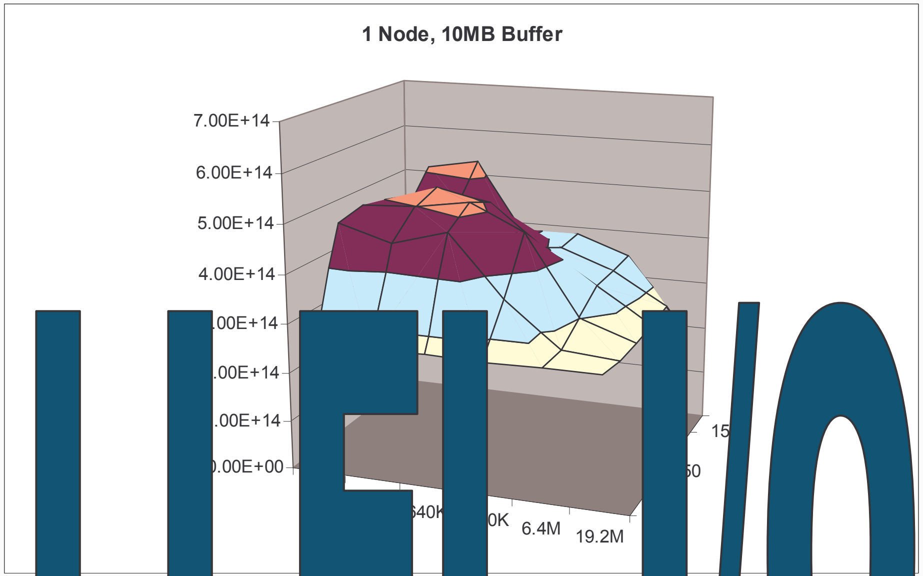
Number of stripes

1	10	50	100	150	160
150MB/s	134MB/s	135MB/s	139MB/s	149MB/s	148MB/s

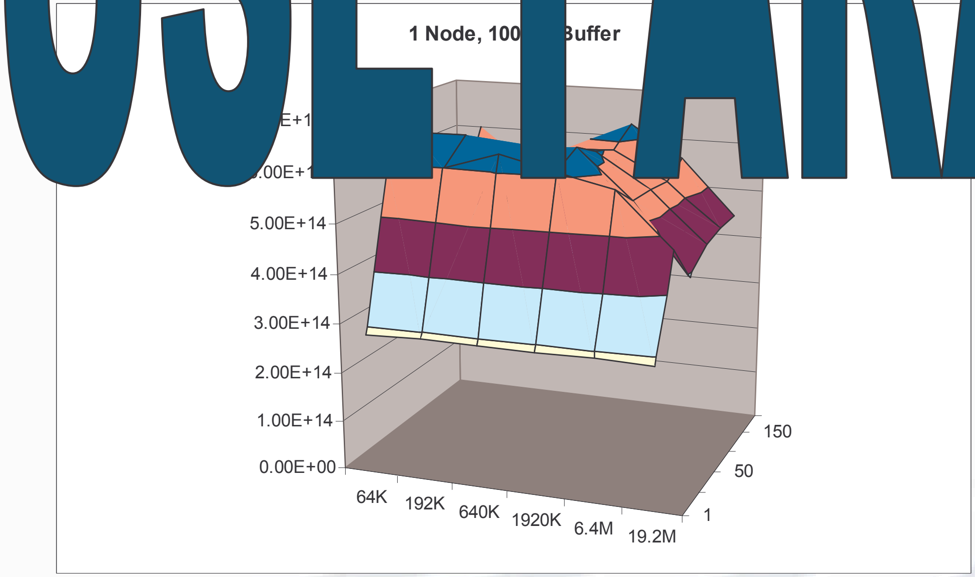
- Using a single IO node:
  - number of stripes doesn't matter
  - stripe size doesn't matter (timings not shown)

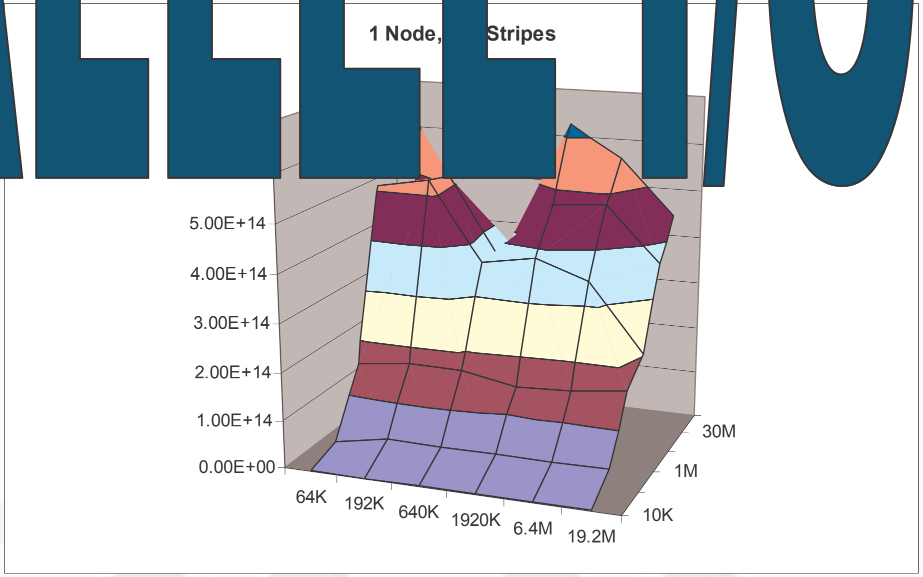
Benchmark Results: 1 I/O Node - Stripes


1 Node, 10K Buffer


1 Node, 10MB Buffer

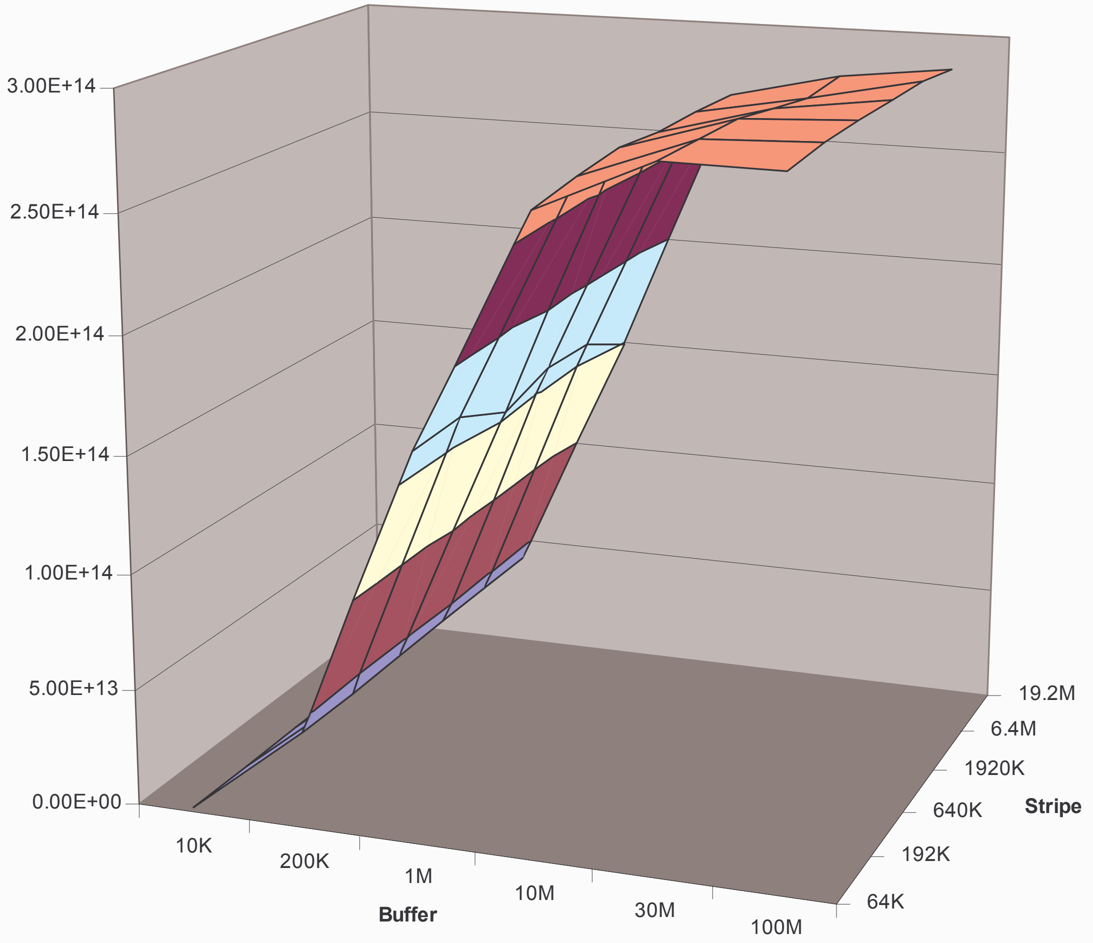
USE PARALLEL I/O!


1 Node, 100 Buffer

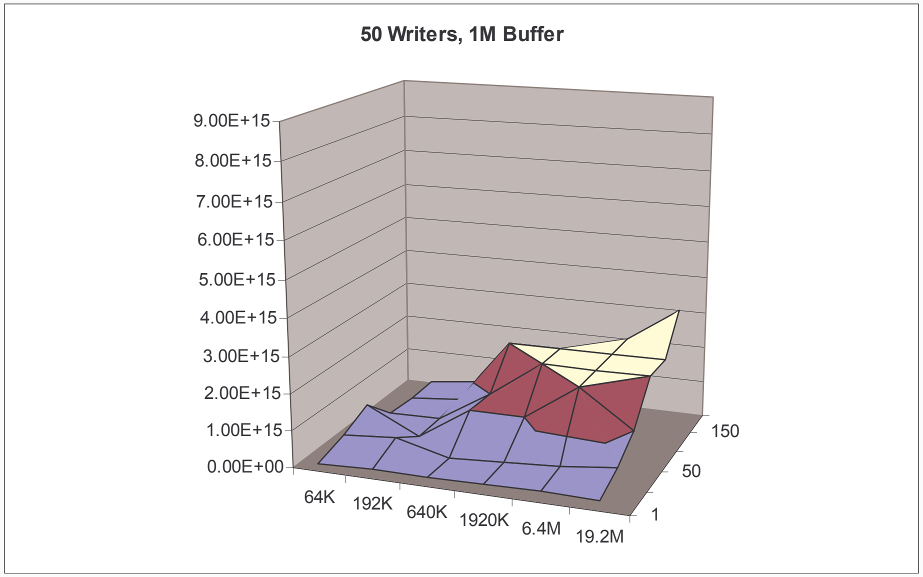

1 Node, Stripes

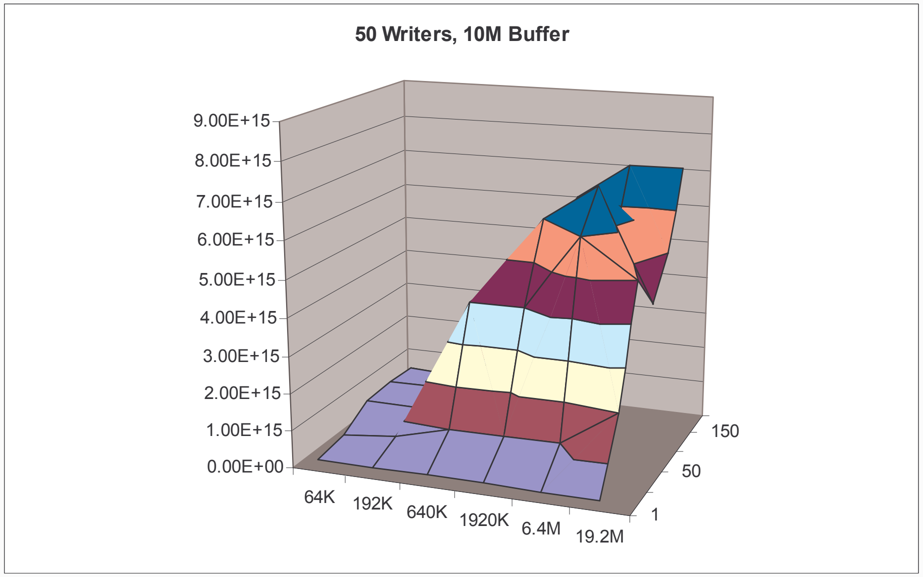
Benchmark Results: 1 I/O Node – Buffer Size

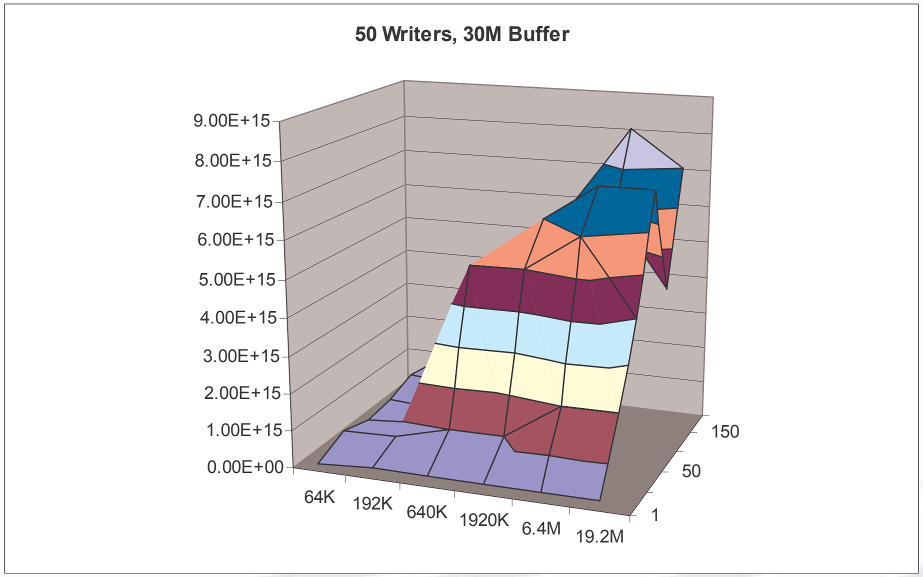
- Single node, single stripe: bandwidth of IO write to disk for different buffer sizes
  - Buffer size is the size of contiguous memory on one IO node written to disk with one write
- Buffer size should be at least 10 megabytes

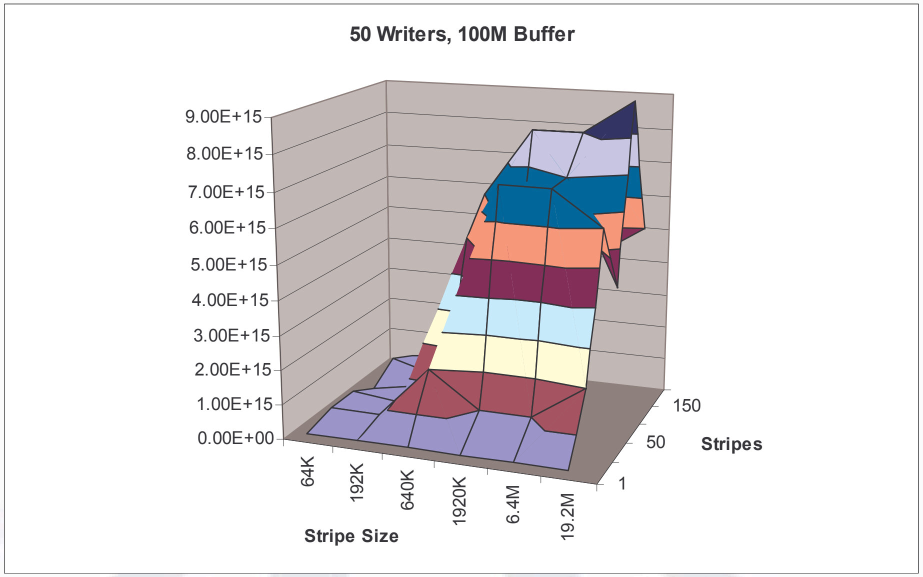


50 Writers, Varying Stripe Count, Size and Buffer Size

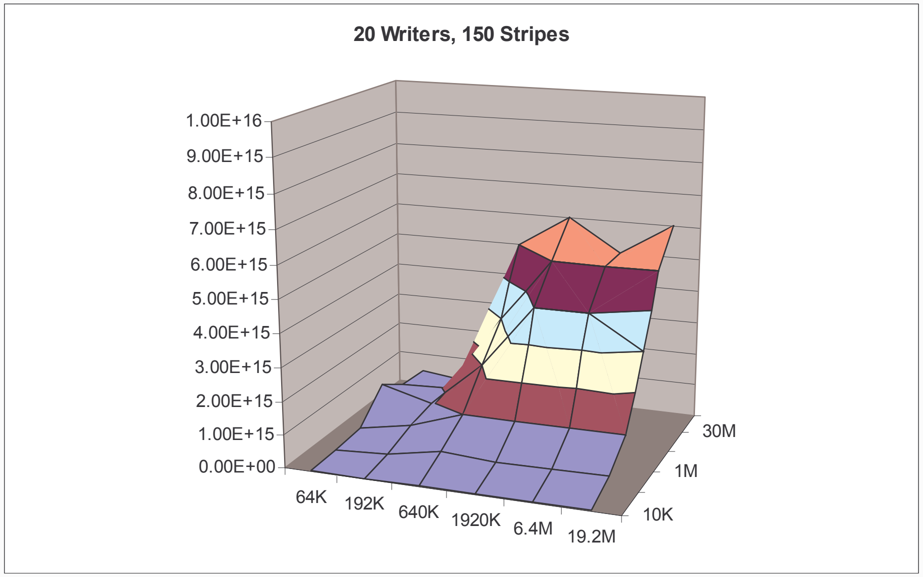

50 Writers, 1M Buffer

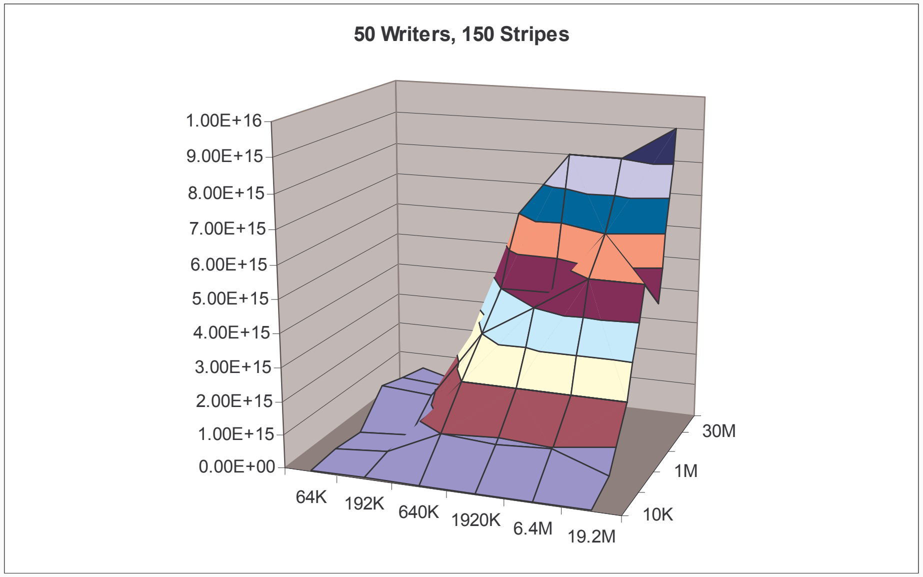

50 Writers, 10M Buffer

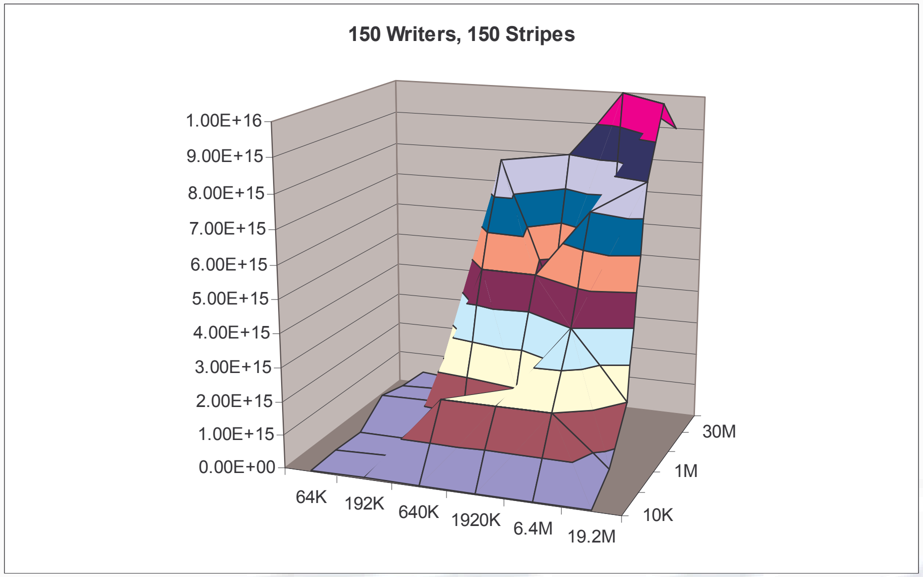

50 Writers, 30M Buffer

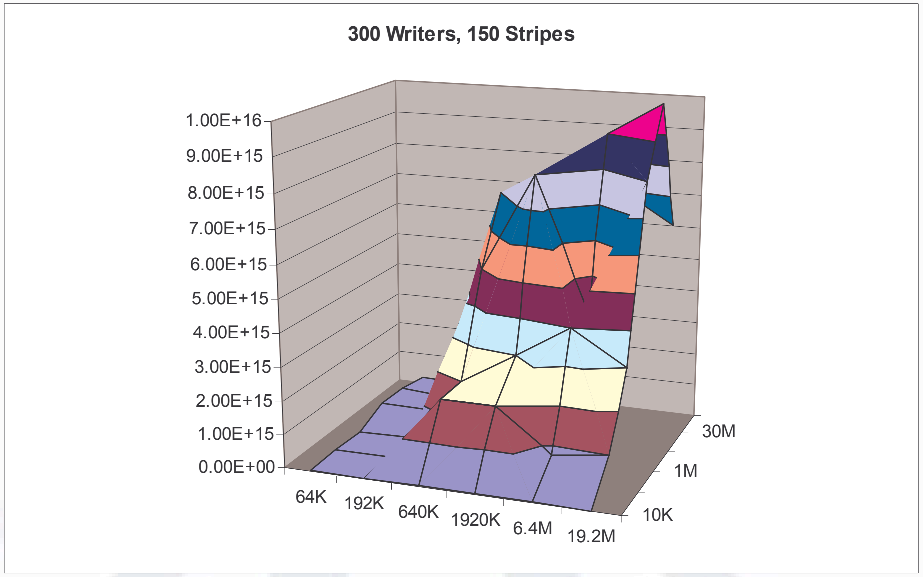

50 Writers, 100M Buffer

150 Stripes, Varying Writers, Buffer, and Stripe Sizes


20 Writers, 150 Stripes


50 Writers, 150 Stripes


150 Writers, 150 Stripes


300 Writers, 150 Stripes

Cliff's Take Home Notes

- Do Large I/O Operations in Parallel MPI-IO
- Create a natural partitioning of nodes so that data will go to disk in a way that makes sense
- Stripe as close to the maximum OSTs as possible given your partitioning
- Use buffers of at least 1MB, 10MB if you can afford it
- Make your I/O flexible so that you can tune to the problem and machine
  - One hard-coded solution will meet your some of the time, but not all of the time
- Come to CUG 2007 and see the application results!